DTIC FILE COPY

Naval Research Laboratory

Washington, DC 20375-5000

NRL Report 9222

AD-A211 343

Design of an Expandable Digital Signal Processor (DSP) Based on the TMS320C25

T. M. MORAN

Human-Computer Interaction Laboratory
Information Technology Division

August 9, 1989

DTIC
ELECTE
AUG17 1989
S D
E

Approved for public release; distribution unlimited.

89 8 17 115

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Report 9222	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Research Laboratory 6b OFFICE SYMBOL (If applicable) Code 5531	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code) Washington, DC 20375-5000	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING/SPONSORING ORGANIZATION Space and Naval Warfare Systems Command 8b OFFICE SYMBOL (If applicable) PMW-151-21	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code) Arlington, VA 22217	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 61153N PROJECT NO: X7290-CC TASK NO: WORK UNIT ACCESSION NO: DN 280-290

11 TITLE (Include Security Classification)
Design of an Expandable Digital Signal Processor (DSP) Based on the TMS320C25

12 PERSONAL AUTHOR(S)
Moran, T. M.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Interim	FROM _____ TO _____	1989 August 9	19

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Signal processor design Real-time speech processor emulation TMS320C25 DSP chip

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Our expandable digital signal processor (DSP) is designed to suit a wide variety of signal processing tasks. It has good flexibility in implementing software because of its unique combination of features. Each DSP has 32K words of program and data EEPROM (electrically erasable and programmable read-only memory), 16K words of program random-access memory (RAM), and 64K words of data RAM. With the capability of remapping external memory by use of software, each microprocessor is capable of running at 10 million instructions per second (MIPS). When more processing power is needed, additional basic DSP units can be added that are capable of communicating with each other through 1K words of global RAM. In addition, each basic unit of the DSP has software-controlled analog I/O circuitry with A/D and D/A conversion rates that can be set independently from 1 to 20 kHz.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Thomas M. Moran	22b TELEPHONE (Include Area Code) (202) 404-7106 22c OFFICE SYMBOL Code 5531

DD Form 1473, JUN 86 *Previous editions are obsolete* SECURITY CLASSIFICATION OF THIS PAGE

CONTENTS

INTRODUCTION 1

BACKGROUND 1

ARCHITECTURE 2

HARDWARE 3

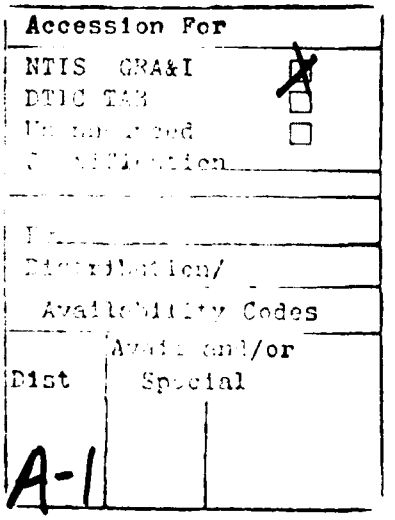
- TMS320C25 Microprocessor 3
- EEPROM 5
- Analog Input/Output 6
- Global Memory 8
- Memory-Select Logic 10
- Wait-State Generator 10
- Backplane and Layout 10

SOFTWARE DEVELOPMENT 12

CONCLUSIONS 14

ACKNOWLEDGMENTS 15

REFERENCES 15



DESIGN OF AN EXPANDABLE DIGITAL SIGNAL PROCESSOR (DSP) BASED ON THE TMS320C25

INTRODUCTION

This report presents the design of an expandable digital signal processor (DSP) developed at the Naval Research Laboratory (NRL) for real-time emulation of digital telephones. The expandable DSP can be composed of a variety of basic DSP units operating in parallel. The basic DSP units are based on the Texas Instruments TMS320C25 microprocessor. Each basic DSP unit is identically equipped with random-access memory (RAM), electrically erasable and programmable read-only memory (EEPROM), global memory buffers, and an analog input and output (I/O) interface. The basic DSP units can operate independently or concurrently through global memory. Important attributes of our design include expandability, high throughput, and compact size.

BACKGROUND

The NRL DSP was developed to emulate various voice-processing algorithms [1] as part of our secure-voice program. Because voice information flows in real time, a newly developed voice algorithm must be tested in real time to determine speech intelligibility, quality, and communicability. Real-time simulation is also useful for early detection of computational pitfalls or undesirable speech effects created by numerical errors. Thus, real-time testing has always been an essential step in the development of voice algorithms (Fig. 1).

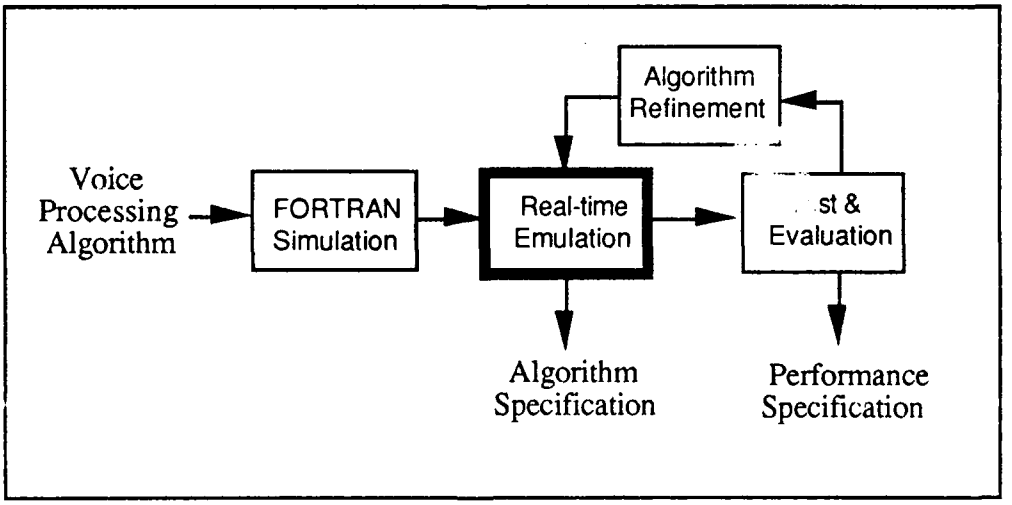


Fig. 1 — Voice algorithm development phases. An engineering prototype (indicated by the thick-lined block) allows us to test a new voice-processing algorithm extensively under various operating conditions. From test results, we can establish the expected performance of the algorithm prior to its specification for production by industry. This procedure has been successfully used for the development of two voice processors currently deployed by the Navy.

Until recently, whenever a new voice-processor was needed, the Navy had industry build special processors because off-the-shelf hardware was not available. For example, the Navy procured a special processor from ITT to emulate the 2400-b/s LPC algorithm used in the Advanced Narrowband Digital Voice Terminal. Similarly, the Navy procured a processor from TRW to emulate the 16,000-b/s linear-predictive-coding algorithm developed for the Navy Secure Conferencing Project and another processor to emulate a very-low-data-rate voice algorithm. At times, and at considerable expense, the Navy has asked industry to generate software based on Navy-developed algorithms.

With the availability of inexpensive and powerful microprocessors made specifically for digital signal processing, we can now design and build our own DSP that is suitable for real-time emulation of voice-processing algorithms. Recently, we developed a DSP that is powerful yet flexible enough to emulate a range of complex voice algorithms.

ARCHITECTURE

Figure 2 is a block diagram of the DSP we implemented to test and demonstrate digital voice-processing algorithms. Each basic unit of the DSP is equipped with a Texas Instruments TMS320C25 microprocessor, RAM to execute its software, EEPROM to store programs and data permanently, and an analog I/O interface for connecting to microphones and speakers. Each DSP unit can run in a stand-alone configuration with the software stored on the EEPROM, or the TMS320C25 microprocessor can be emulated with the software stored on a host computer.

Basic DSP units, whether powered by TMS320C25 microprocessors or emulators, can operate in parallel while communicating through a global memory. Global memory is attached to each microprocessor via the global memory bus. Buffers reside on each microprocessor board to connect each basic DSP unit's data and address buses to the global memory bus. Since only one microprocessor can communicate with global memory at a time, arbitration of the bus is decided by logic circuitry residing on the global memory board.

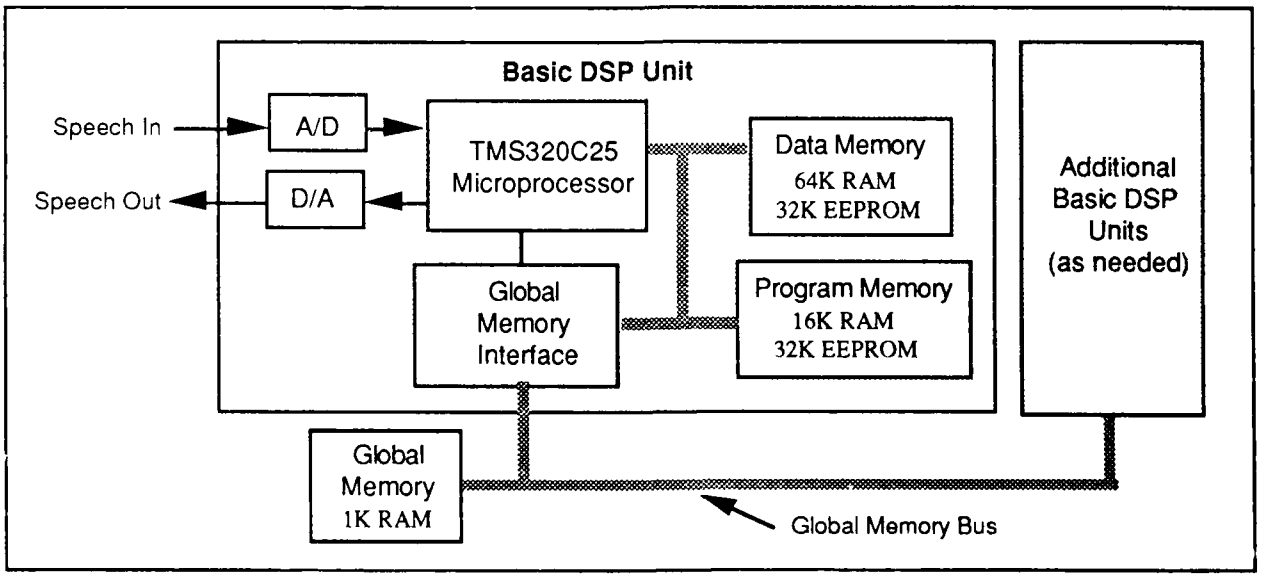


Fig. 2 — DSP designed and fabricated at NRL to emulate voice-processing algorithms. The ability of basic DSP units to operate in parallel makes this a powerful yet flexible computer. This parallel processing ability allows the computer to accommodate complex algorithms well into the future by expanding to fill our processing needs.

The driving forces of this architecture are expandability and flexibility. It is difficult to predict future computational needs, so it is desirable to be able to increase the processing power as it becomes necessary. Since the microprocessor is already being run at its maximum speed, greater processing capacity is achieved by partitioning the algorithm into subprograms that can process in parallel. Many voice-processing algorithms are, by their nature, suitable for partitioning. These different subprograms can be distributed among enough processors to achieve real-time operation. The advantage of our DSP architecture is that basic DSP units can be added until adequate processing capacity is achieved. The DSP hardware configuration would include as many basic DSP units as needed to implement the algorithm in real time. Our prototype DSP is capable of having three basic DSP units running in parallel; the only limit to the number of these units operating together is based on timing, which is discussed later.

HARDWARE

Physically, each basic DSP unit is composed of two circuit boards—the microprocessor board and the EEPROM board. The microprocessor board contains the TMS320C25 microprocessor, 64K of data RAM, 16K of program RAM, buffers for the global memory bus, a 40 MHz oscillator, memory select logic, and a wait-state generator. The EEPROM board contains 32K of program and 32K of data EEPROM and the analog I/O circuitry. Basic DSP units operating in parallel (Fig. 2), communicate with each other by global memory that is resident on a separate circuit board.

All the circuit boards of the DSP have the same physical dimensions. Each board has a 122-pin edge connector that fits into the backplane of the DSP's cardcage. The edge connector physically secures the circuit board into the cardcage while providing signal, power, and ground connections. The two circuit boards comprising the basic DSP unit are paired interchangeably in the cardcage.

The TMS320C25 microprocessor, data and program RAM, data and program EEPROM, and the global memory buffers are all directly attached to the address and data buses. The analog I/O circuitry consists mainly of a Texas Instruments TLC32044 codec that provides analog-to-digital (A/D) conversion, digital-to-analog (D/A) conversion, and filtering on one chip. The address bus, data bus, and analog I/O bus between the two circuit boards are connected through the backplane of the cardcage.

TMS320C25 Microprocessor

The TMS320C25 microprocessor is a very-large-scale-integration (VLSI) chip that is capable of multiplying two 16-bit numbers into a 32-bit accumulator in a single instruction cycle time of 100 ns. Thus, the TMS320C25 microprocessor is ideally suited for computation-intensive applications where vector products are often carried out. These products include speech processing, spectral analysis, modulation/demodulation, digital filtering, convolution/correlation operations, and graphic/image processing. In addition, the A/D and D/A converters can be interfaced with the TMS320C25 microprocessor through either a serial or a parallel port (Fig. 3).

Besides having 544 words of on-chip RAM, the TMS320C25 microprocessor is capable of addressing up to 64K words of external data memory and 64K words of external program memory. The TMS320C25 microprocessor provides the necessary control lines for interfacing with other processors. Also, the software has the capability to allocate global data memory in place of its local data memory. Once global memory has been allocated, the storage or retrieval of data to either local or global data memory is software transparent.

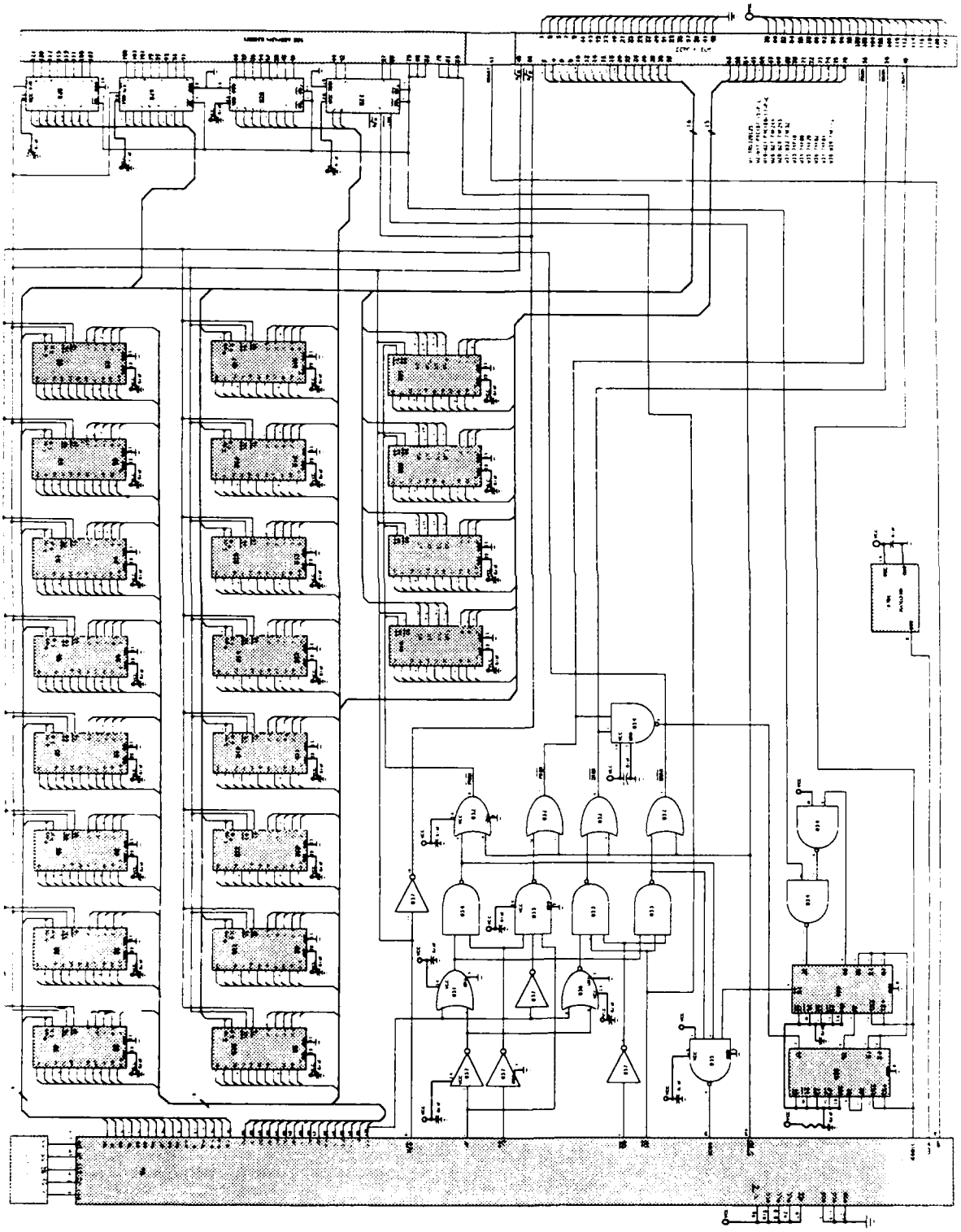


Fig. 3 — Processor board schematic. This is one of two circuit boards needed for each basic DSP unit; it contains a TMS320C25 microprocessor, program and data RAM, global memory buffer, memory-select circuitry, and a wait-state generator.

EEPROM

The use of EEPROM extends the flexibility of the DSP and simplifies the algorithm development process. Each basic DSP unit has 32K of EEPROM residing in program memory space and 32K of EEPROM residing in data memory space. By storing software in nonvolatile EEPROM, the DSP can operate without being attached to a host. This stand-alone capability enables the DSP to be portable and permits more effective algorithm testing and demonstration.

The EEPROM eliminates the need for an emulator for each microprocessor board during software development. Software can be developed in stages on a host computer and tested on an emulator. When each section of the software is completed, it can be downloaded to the EEPROM of one of the basic DSP units. The next part of the software can then be tested on the emulator while the first section of software is running on the basic DSP unit. As needed, more units can be added and more EEPROM can be programmed; software in EEPROM can be reprogrammed quickly and can be reprogrammed many times.

EEPROM is programmed with software stored on a host computer. The software is downloaded from the host computer to RAM prior to being transferred to EEPROM. Therefore, the RAM and the EEPROM need to be attached to the emulator; this is done by disconnecting the TMS320C25 microprocessor from each basic unit and connecting the emulator plug in its place. This creates some problems because removing the emulator plug requires considerable effort and risks damaging the delicate pins. These problems are compounded by the frequent need to make changes to the software during the development process.

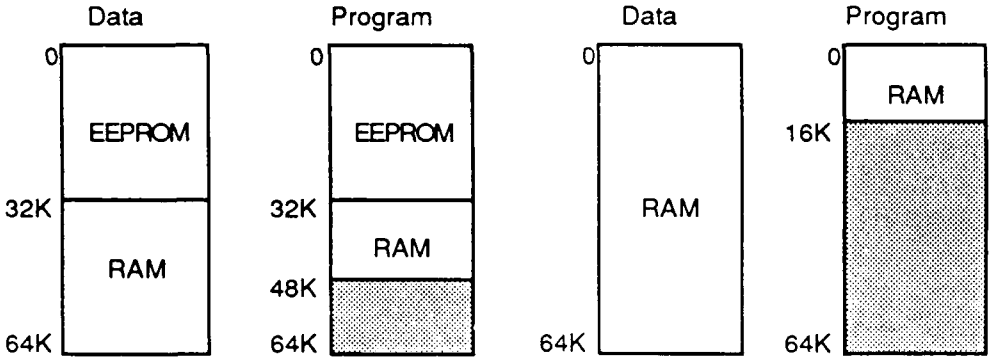
By having EEPROM on a physically separate board, programming the EEPROM is accomplished faster, with less physical stress to the components. The emulator remains plugged into a dedicated microprocessor board that has the same RAM and bus connections as the other microprocessor boards. When EEPROM is to be programmed, the EEPROM board is fitted in the cardcage alongside the emulator board that connects the EEPROM through the backplane to the emulator. After programming and testing, the EEPROM board can then be moved into a slot alongside a microprocessor board where it is likewise connected.

Although the voice-processing software is stored in EEPROM, the slow access time prohibits actually running the software from the EEPROM. Because a microprocessor requires four wait states for each memory access when running software from EEPROM, the effective speed of the DSP is greatly reduced. To run the microprocessor at full speed, no wait states can be initiated on a memory access. Because only RAM is fast enough to run without the need for wait states, software stored in EEPROM is copied to RAM before processing begins.

Transferring software from EEPROM to RAM causes some problems. These problems are resolved by the capability of swapping EEPROM and RAM. Moving data, especially in table form, from EEPROM to RAM presents little trouble. When data are copied into RAM, the main concern is whether, afterwards, an adequate amount of RAM is available for use by the software. Remapping memory solves this problem by placing another 32K of RAM into data memory space, for a total of 64K of data RAM. Copying programs into different memory locations may interfere with subroutine addressing. Since address changes must be accounted for within the program, the need for programs to be moved around in memory space can make programming very complex. This difficulty is also overcome by remapping memory. Because program RAM maps over the same addresses as program

EEPROM, programs copied from EEPROM to RAM before remapping remain in the same memory locations after remapping. Because the address of the program does not really change (only the device the program resides in changes), software development is greatly simplified.

When the DSP is turned on, the memory map is in the state shown in Fig. 4(a). Utility software copies the voice-processing software from EEPROM to RAM. When the process is complete, the memory space occupied by EEPROM is remapped. EEPROM is replaced with RAM containing the voice-processing software (shown by memory map Fig. 4(b)). The software is then run from the memory address space newly occupied by RAM. Thus, RAM is accessed without any wait states; the microprocessor runs at full speed, and no processing capacity is sacrificed.



(a) Data memory address locations 0 to 32K are occupied by EEPROM, and 32K to 64K are occupied by RAM. Program locations 0 to 32K are occupied by EEPROM, and 32 to 48K are occupied by RAM. Program locations 48K to 64K are empty.

(b) The entire 64K words of data memory locations are occupied by RAM. Program locations 0 to 16K are occupied by RAM (the same RAM that resides at 32K to 48K in Fig. 4(a)). Program locations 16K to 64K are empty.

Fig. 4 — External memory map. Shaded areas imply nonexisting memories.

There are other advantages for being able to remap EEPROM and RAM by software control. One advantage, resulting from the increased storage capacity, is the ability to switch between different algorithms. Depending on their size, several algorithms can be stored in EEPROM. Each algorithm can be tailored to different environmental situations and can be copied into RAM as external situations change. Similarly, algorithms can actively jump between different tables of data stored on EEPROM. Figure 5 is a schematic of the EEPROM board.

Analog Input/Output

A Texas Instruments TLC32044C provides all the necessary analog I/O functions on one chip. It has a 14-bit A/D converter, a 14-bit D/A converter, a bandpass antialiasing input filter, a lowpass output filter, internal reference voltage, programmable sample/conversion rates up to 20 kHz, and a serial port interface. The serial port has four modes; one of these modes, the byte mode, provides a direct interface to the TMS320C25 when operating asynchronously. Another feature, which is useful for certain speech algorithms, is the ability to set the A/D and D/A conversion rates independently when the TLC32044C is operated asynchronously. This chip, together with the two analog I/O connectors, resides on the same board as the EEPROM (Fig. 5). It has a separate analog power supply

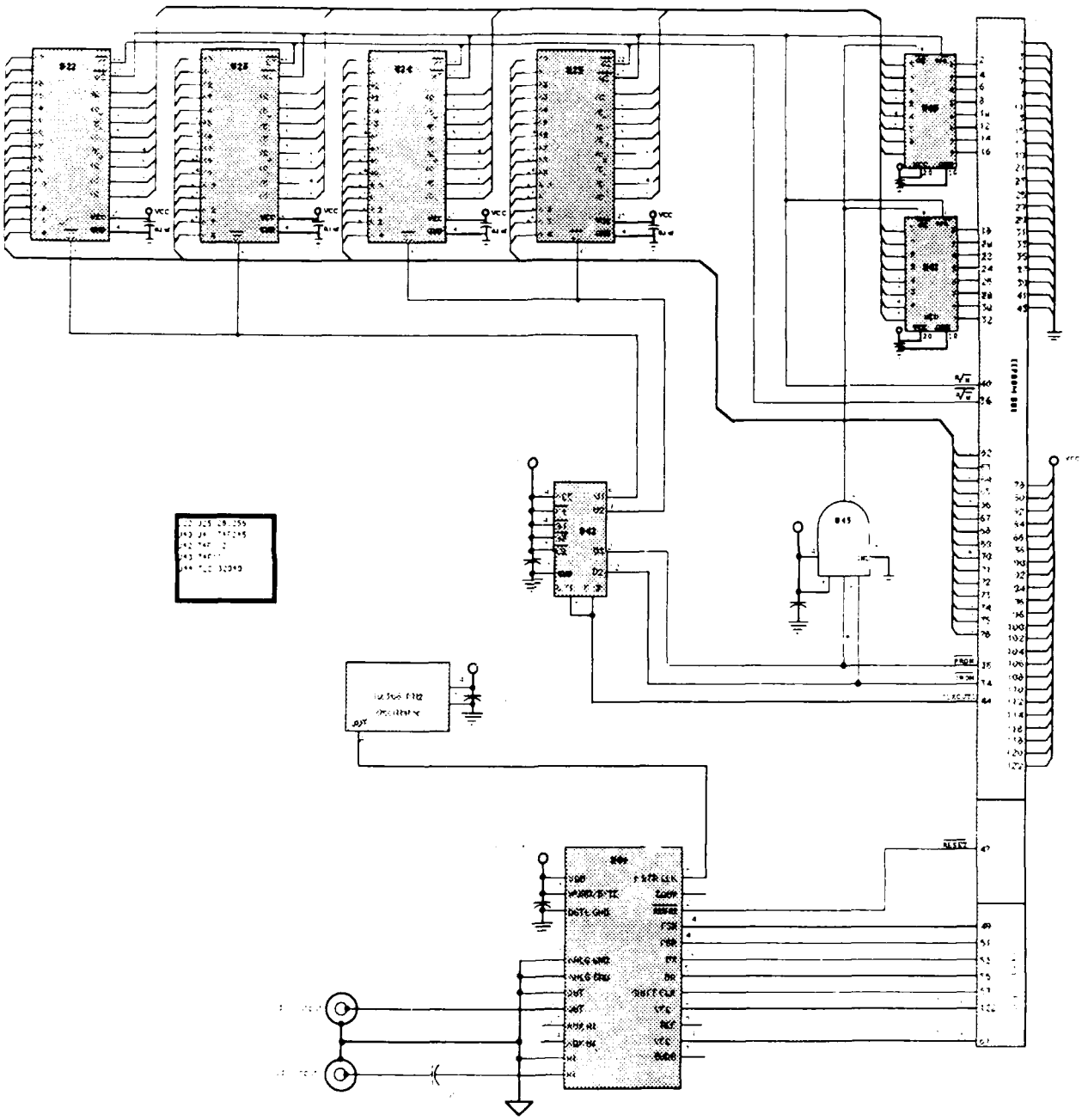


Fig. 5 — EEPROM and analog I/O board, one of two boards necessary for the basic DSP unit

that is filtered on this board at the input to the chip. The analog ground is shared with the digital ground, but they are connected at only one point to reduce the possibility of noise.

Global Memory

To run in parallel, each microprocessor must be able to communicate with the others. The TMS320C25 is designed to do this by sharing global memory (i.e., memory that can be accessed by all microprocessors). Each microprocessor can map global memory onto its own data memory space with a software command. Afterwards, global-memory access is software transparent. The microprocessors share 1K words of global memory (Fig. 6).

Each microprocessor, by being part of a basic DSP unit, is connected to global RAM through the global memory bus. Since only one microprocessor can use the bus at a time, buffers between global and local memories reside on each microprocessor board. Arbitration for the global memory bus is done asynchronously by logic on the global memory board. This logic sequentially checks the bus-request lines coming from the microprocessors.

When a bus-request signal is detected from a microprocessor wanting access to global memory, arbitration logic returns an acknowledge signal to that microprocessor board. The microprocessor making the request is in a wait state until the bus-request signal is acknowledged. After acknowledgment, the microprocessor comes out of the wait state and gains control of the global memory bus. It has access to the global memory for one instruction cycle. At the end of the cycle, control of the global memory is relinquished by the microprocessor, and the arbitration logic begins to check the other bus-request lines.

By arbitrating the use of the global memory bus asynchronously, each microprocessor has an equal opportunity to gain access to global memory. Since the duration for a global access is controlled by that microprocessor's instruction cycle time, microprocessors running at different speeds can still share the same memory and still be able to run in parallel.

The disadvantage of the asynchronous arbitration, compared to a synchronous arbitration scheme, is the extra time needed, on average, to gain access to global memory. The arbitration logic sequentially cycles through the bus request lines. A microprocessor must wait its turn for a request to be detected and acknowledged, even if no other requests are being made. But, as in any situation involving a shared bus, a microprocessor must wait if another microprocessor is already controlling global memory. Of course, more microprocessors attempting to use global memory at the same time require a longer waiting period for each.

The advantage of this asynchronous design is that all microprocessors have equal priority (no microprocessor can lock up global memory, and no microprocessor needs to wait longer than any other). No special software is needed to prevent access to global memory by the other microprocessors.

Our global-memory arbitration circuitry allows up to three basic DSP units to be attached to global memory, but the design is open-ended. Additional DSP units can be accommodated by expanding this circuitry. The penalty for expanding is an increase of the average time for each microprocessor to make a global-memory access.

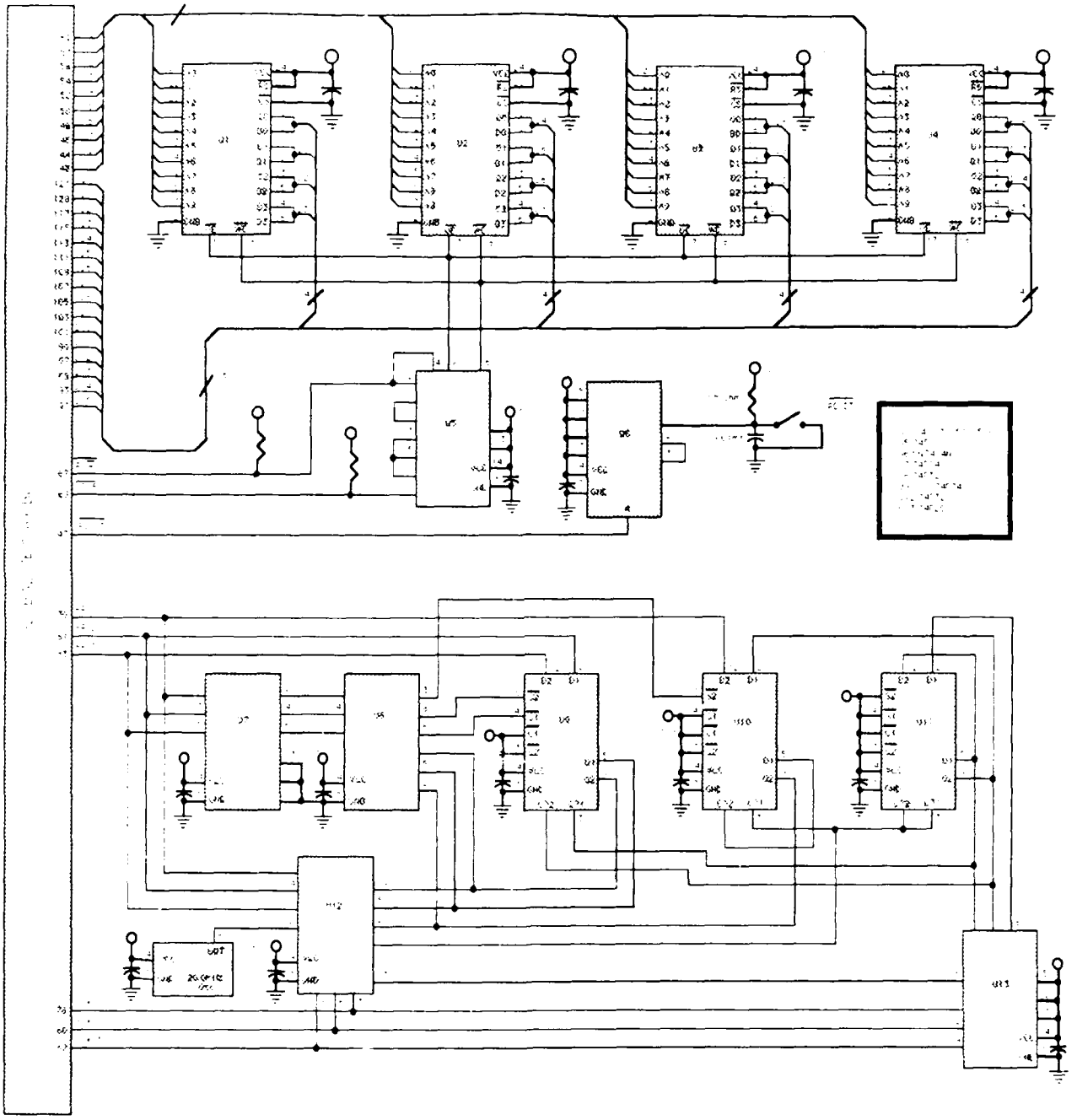


Fig. 6 — Global memory board schematic showing global RAM, arbitration circuit, and reset circuitry. This board is required only when the basic DSP units are operating in parallel.

Memory-Select Logic

Memory-select logic (shown on the left-hand side of Fig. 3) decodes the signals coming from the microprocessor and selects the appropriate memory chips. Memory chips to be selected include program RAM, data RAM, program EEPROM, data EEPROM, and global RAM. In addition, logic signals the microprocessor when wait states are needed and initiates the wait-state generator.

Memory-select logic provides the DSP units with the capability of remapping program and data memory spaces. Memory-select logic determines how the memory is mapped by the signal on the microprocessor's software-controllable output pin. The signal on this pin, in conjunction with the address of memory being accessed, controls whether a RAM or an EEPROM device is selected.

Timing restraints play an important part in the design of memory-select logic. The memory devices need to have the address lines active before data are written. When data are being read, the memory chips must be shut down well before the next instruction cycle begins. This prevents the possibility of the memory devices driving the data bus too long, which would cause a conflict with the microprocessor. Meeting these criteria is a critical function of the memory-select logic.

Wait-State Generator

The wait-state generator (shown in the lower left-hand side of Fig. 3) allows the microprocessor to communicate with the slower EEPROM and global memory by extending the length of the microprocessor's memory access. The microprocessor's memory access time is designed to be extended by the initiation of what is called a wait state. A wait state is a condition in which the microprocessor drives the signals to its external memory devices for an extra instruction cycle.

When the wait-state generator, through memory-select logic, detects the microprocessor beginning an access to slow memory, the wait-state generator signals the microprocessor to go into a wait-state. After the wait-state generator stops signaling for wait states, the memory access is completed by the microprocessor. Wait states can occur consecutively, as in the case of an access to EEPROM that requires four consecutive wait states. The program and data RAM run fast enough that no wait states are needed. RAM on the global-memory board is actually fast enough to run without any wait states, but gaining control of the global memory bus requires at least one, and usually several, wait states.

The wait-state generator provides wait states for two conditions. The first condition is when an EEPROM device is selected; as previously stated, four wait states are induced in the microprocessor. The second condition requiring wait states is a global-memory access. During a global-memory access, the microprocessor must wait an indeterminate amount of time for control of global memory to be granted. Since the number of wait states needed is unknown, wait states are generated from the time a global bus request is made until after the acknowledge signal from the global-memory arbitration circuit is received. After the acknowledge signal is received, one extra wait state is generated to ensure adequate address set-up time for the global memory.

Backplane and Layout

Because our DSP is a [illegible] type, the circuits have all been built on wirewrap circuit boards. Any future implementations [illegible]ld be laid out on printed circuit boards, but the configuration would probably be the same. A single DSP unit resides on two circuit boards—a microprocessor board and an EEPROM board. These two boards are paired together to form each basic DSP unit (Fig. 7).

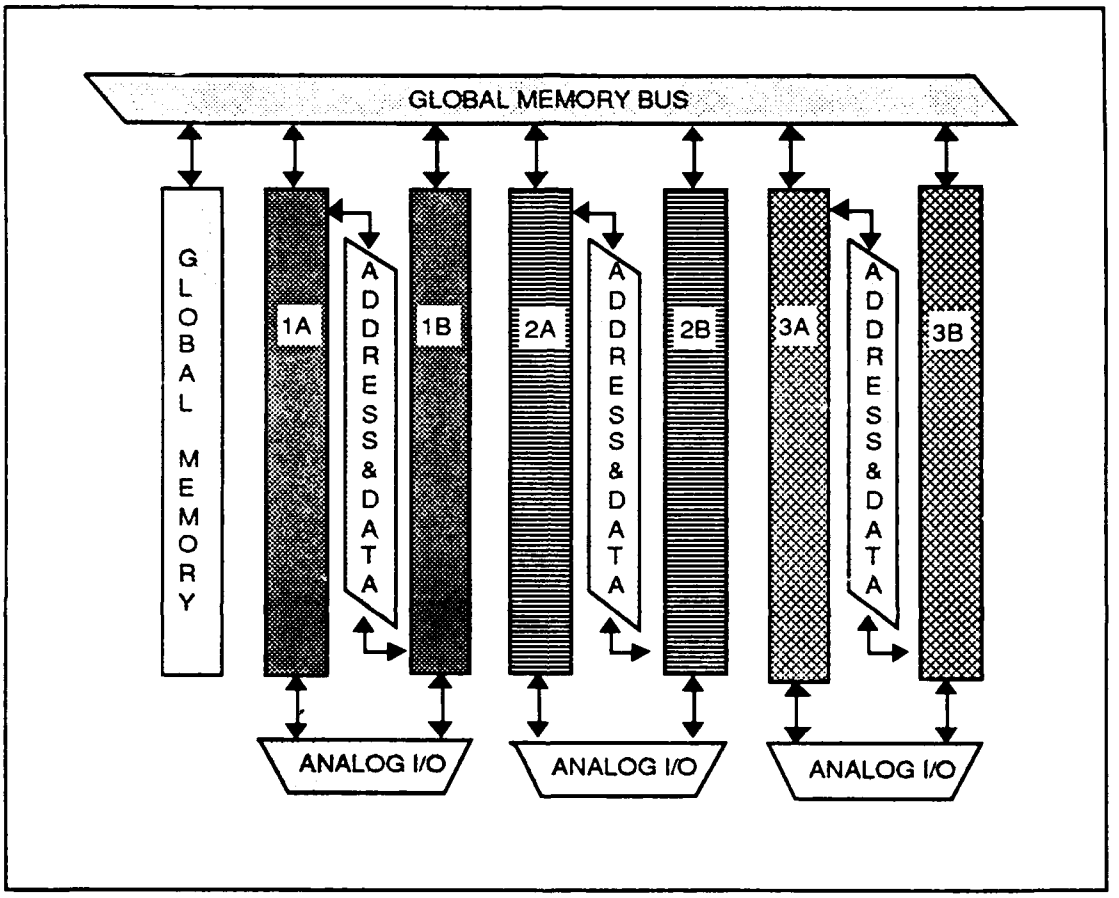


Fig. 7 — Backplane of DSP (rear view) showing the connections on the backplane of a DSP in which three processors are operating in parallel (Fig. 8 shows the fabricated DSP). The microprocessor boards and EEPROM boards may occupy either an A or a B slot, but they must be placed in pairs. When using an emulator, the emulator board may occupy any slot except the global memory position (see Fig. 9).

Fig. 8 — Two basic DSP units and the global-memory board

When operating in parallel, a global-memory board must be added. The global-memory board must be inserted into the first slot of the cardcage. Remaining circuit-board slots are wired together as pairs.

Three separate sets of connections are on the backplane of the cardcage. They are the global-memory bus, address and data buses, and analog I/O connections (Fig. 7). The address and data buses and analog I/O connections join the microprocessor board/EEPROM board pairs. Because the global-memory bus and wires for power and ground extend to every slot in the cardcage, microprocessor boards can be inserted alongside EEPROM boards in either position within the pair. It does not matter which pair of slots the boards occupy. In addition, vacant pairs of slots do not affect operation.

SOFTWARE DEVELOPMENT

The software development process requires frequent testing and rewriting. Development services are provided by a host computer and the TMS320C25 emulator (Fig. 9). Our host computer, a VAX 11/780, provides the utility resources necessary to write, compile, and store the DSP's software (Table 1 is an example of software generated for the DSP described in this report). When the machine language software is ready to be tested, it is downloaded from the host, through the emulator, to the emulator board's RAM. The software can then be run with the emulator. For more extensive testing and for parallel operation, the software can be run on a microprocessor by using the emulator to oversee operations through the global memory.

For parallel operation, software must be written and tested in segments. Although it is possible to attach a separate emulator to each DSP unit, using multiple emulators is not always necessary or

Table 1 — An Example of Software Written for the DSP. This particular software was developed for removing undesirable DC bias created by the A/D converter resulting from component aging. This DC filter has a zero at $z = 1$ and pole at $z = 31/32$. Thus, the transfer function is $H(z) = (1 - z^{-1}/[1 - (31/32)z^{-1})]$. This software was written by Larry Fransen of NRL.

```
*************************************************************
*                                                           *
*       DC REMOVER                                          *
*                                                           *
*               DIMENSION A2DBUF(0:IFRMX2-1)                *
*               DIMENSION XDCGON(0:IFRMX4-1)                *
*                                                           *
*               DO 50 I=0,IFRMX3-1                          *
*       50      XDCGON(I)=XDCGON(I+IFRAME)                  *
*                                                           *
*               J=IFRAME                                    *
*               IF(XBUFSW.EQ.0) J=0                         *
*               K=0                                         *
*               DO 100 I=0,IFRAME/2-1                       *
*               XDC1B=A2DBUF(J)                             *
*               XDC2=(A2DBUF(J)-XDC1A)+(31/32)*XDC2         *
*               XDCGON(K)=XDC2                              *
*               J=J+1                                       *
*               K=K+1                                       *
*               XDC1A=A2DBUF(J)                             *
*               XDC2=(A2DBUF(J)-XDC1B)+(31/32)*XDC2         *
*               XDCGON(K)=XDC2                              *
*               J=J+1                                       *
*               K=K+1                                       *
*       100     CONTINUE                                    *
*                                                           *
*               RETURN                                      *
*                                                           *
*************************************************************
*
DC      SOVM                    *SET OVERFLOW MODE
        SPM 3                   *SHIFT MULTIPLIER RIGHT 6 PLACES
        SSXM                    *SET SIGN EXTENSION MODE
*
        LARP 1                  *ARP=1
        LRLK 0,XDCGON           *AR0 PTS TO XDCGON(0)
        LRLK 1,XDCGON+IFRAME    *AR1 PTS TO XDCGON(IFRAME)
        LRLK 2,IFRMX3-1         *AR2=IFRMX3-1
DCAA    LAC *+,0,0              *ACC=XDCGON(I+IFRAME),ARP=0
        SACL *+,0,2             *XDCGON(I)=XDCGON(I+IFRAME),ARP=2
        BANZ DCAA,*-,1          *BRANCH,ARP=1
*
        LRLK 1,XBUFSW           *AR1 PTS TO XBUFSW
        LAC *,0,0               *ACC=XBUFSW,ARP=0
        BNZ DCA                 *BRANCH
        LRLK 0,A2DBUF           *AR0 PTS TO A2DBUF(0),FIRST BUFFER
        B DCB                   *BRANCH
DCA     LRLK 0,A2DBUF+IFRAME    *AR0 PTS TO A2DBUF(IFRAME),SECOND BUFFER
DCB     LRLK 2,XDCGON+IFRMX3    *AR2 PTS TO XDCGON(IFRMX3)
        LARK 3,IFRAME/2-1       *AR3=IFRAME/2-1
        LALK >7BFF              *ACC=31/32,{Q15}
        SACL TMP1               *TMP1=31/32
        LT TMP1                 *T=31/32
DC1     LAC *+,9,2              *ACC=A2DBUF(.),{Q9},ARP=2
        SACH XDC1B,7            *XDC1B=A2DBUF(.)
        SUB XDC1A,9             *ACC=A2DBUF(.)-XDC1A,{Q9}
        MPY XDC2                *P=(31/32)*XDC2,{Q15}
        APAC                    *ACC=(A2DBUF(.)-XDC1A)+(31/32)*XDC2,{Q9}
        SACH XDC2,7             *XDC2=ACC
        SACH *+,7,0             *XDCGON(.)=ACC,ARP=0
        LAC *-,9,2              *ACC=A2DBUF(.),{Q9},ARP=2
        SACH XDC1A,7            *XDC1A=A2DBUF(.)
        SUB XDC1B,9             *ACC=A2DBUF(.)-XDC1B,{Q9}
        MPY XDC2                *P=(31/32)*XDC2,{Q15}
        APAC                    *ACC=(A2DBUF(.)-XDC1B)+(31/32)*XDC2,{Q9}
        SACH XDC2,7             *XDC2=ACC
        SACH *+,7,3             *XDCGON(.)=ACC,ARP=3
        BANZ DC1,*-,0           *BRANCH,ARP=0
*
        SPM 0                   *NO MULTIPLIER SHIFT
        RET                     *RETURN
```

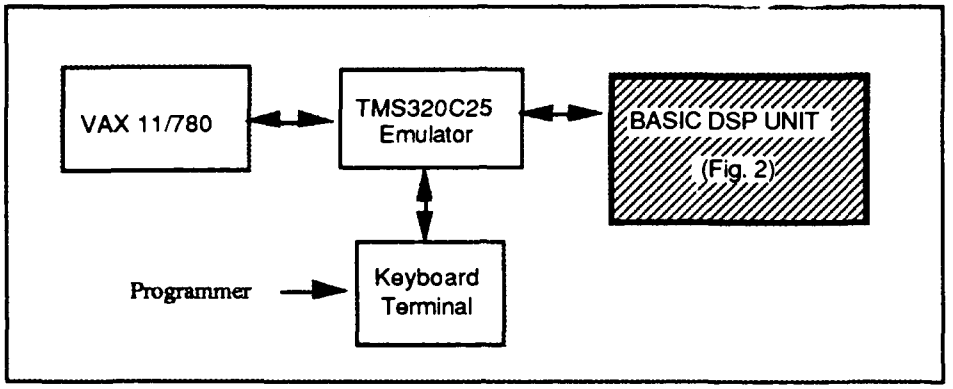


Fig. 9 — The basic DSP unit operated with an emulator and host computer. Although the basic DSP unit shown in Fig. 2 can run as an independent computer, it can also be operated with the host computer and emulator as shown here. The signal-processing software is generated on the host computer where it can be altered and stored easily. This setup also allows the programmer to monitor the execution of software and hardware.

desirable. As each segment is completed it can be stored in EEPROM and be run on a DSP unit, independent of and in parallel with the host computer.

Independent operation of basic DSP units, whether the units are in parallel or alone, requires two utility programs. The first utility program, the EEPROM loading program, controls the process of writing software to EEPROM. The extra long timing requirements of an EEPROM write cycle preclude simply downloading the software to EEPROM as is done with RAM. The timing requirements are better handled with software, rather than with hardware. The EEPROM loading program provides the extra timing delays and data validation necessary when writing to EEPROM. The second utility program, the boot program, is required by basic DSP units to operate from RAM when running independently. The boot program copies the software from EEPROM to RAM just after power is applied to the DSP. Both the EEPROM loading program and the boot program are attached to and compiled with the signal processing program on the host computer. These three programs are downloaded as one program to the emulator board's program RAM. When run, the loading program, through the emulator, copies the boot program and the signal processing program to EEPROM. After these two programs are placed in EEPROM, the EEPROM board can be placed alongside the microprocessor board. The boot program will, at power-up, automatically place the signal processing software into RAM, and the program will begin running.

CONCLUSIONS

Our expandable DSP is designed to suit a wide variety of signal-processing tasks. It has good flexibility in implementing software because of its unique combination of features. Each DSP has 32K words of program and data EEPROM, 16K words of program RAM, and 64K words of data RAM. With the capability of remapping external memory by software, each microprocessor is capable of running at 10 MIPS. When more processing power is needed, additional basic DSP units can be added that are capable of communicating with each other through 1K words of global RAM. In addition, each basic unit of the DSP has software-controlled analog I/O circuitry with A/D and D/A conversion rates that can be set independently from 1 to 20 kHz.

ACKNOWLEDGMENTS

The author thanks Sharon James, Howard Brown, and Tim McChesney of the Space and Naval Warfare Systems Command for support of these development projects. The author also thanks Georg Thomas of NRL who initiated this project, Larry Fransen who tested the fabricated DSP while generating voice-processing software, Dave Tate for his advice, and George Kang for his help in putting it all together.

REFERENCES

1. G.S. Kang and L.J. Fransen, "Low-Bit Rate Speech Encoders Based on Line-Spectrum Frequencies (LSFs)," NRL Report 8857, Jan. 1985.

2. "TMS320C25 User's Guide," Texas Instruments Incorporated, P.O. Box 1443, Houston, TX 77001 (1986).

3. "Digital Signal Processing Applications with the TMS320 Family," Texas Instruments Incorporated, P.O. Box 1443, Houston, TX 77001 (1986).